FatNemo: Building a Resilient Multi-source Multicast Fat-Tree

Stefan Birrer, Dong Lu, Fabián E. Bustamante, Yi Qiao, and Peter Dinda

Northwestern University, Evanston IL 60201, USA,
{sbirrer,donglu,fabianb,yqiao,pdinda}@cs.northwestern.edu

Abstract. This paper proposes the idea of emulating fat-trees in overlays for multi-source multicast applications. Fat-trees are like real trees in that their branches become thicker the closer one gets to the root, thus overcoming the “root bottleneck” of regular trees. We introduce FatNemo, a novel overlay multi-source multicast protocol based on this idea. FatNemo organizes its members into a tree of clusters with cluster sizes increasing closer to the root. It uses bandwidth capacity to decide the highest layer in which a peer can participate, and relies on co-leaders to share the forwarding responsibility and to increase the tree’s resilience to path and node failures.

We present the design of FatNemo and show simulation-based experimental results comparing its performance with that of three alternative protocols (Narada, Nice and Nice-PRM). These initial results show that FatNemo not only minimizes the average and standard deviation of response time, but also handles end host failures gracefully with minimum performance penalty.

1 Introduction

High bandwidth multi-source multicast among widely distributed nodes is a critical capability for a wide range of important applications including audio and video conferencing, multi-party games and content distribution. Throughout the last decade, a number of research projects have explored the use of multicast as an efficient and scalable mechanism to support such group communication applications. Multicast decouples the size of the receiver set from the amount of state kept at any single node and potentially avoids redundant communication in the network.

The limited deployment of IP Multicast [16,17], a best effort network layer multicast protocol, has led to considerable interest in alternate approaches that are implemented at the application layer, using only end-systems [14,24,19,32,11,2,10,33,39,31,35]. In an end-system multicast approach participating peers organize themselves into an overlay topology for data delivery. Each edge in this topology corresponds to a unicast path between two end-systems or peers in the underlying Internet. All multicast-related functionality is implemented at the peers instead of at routers, and the goal of the multicast protocol is to construct and maintain an efficient overlay for data transmission.

Among the end-system multicast protocols proposed, tree-based systems have proven to be highly scalable and efficient in terms of physical link stress, state and control overhead, and end-to-end latency. However, normal tree structures have two inherent problems:

C.-H. Chi, M. van Steen, and C. Wills (Eds.): WCW 2004, LNCS 3293, pp. 182–196, 2004.

Resilience: They are highly dependent on the reliability of non-leaf nodes. Resilience is particularly relevant to the application-layer approach, as trees here are composed of autonomous, unpredictable end systems. The high degree of transiency of the hosts [1] has been pointed out as one of the main challenges for these architectures [4].

Bandwidth limitations: They are likely to be bandwidth constrained [2] as bandwidth availability monotonically decreases as one descends into the tree. The bandwidth limitations of normal tree structures is particularly problematic for multi-source, bandwidth intensive applications. For a set of randomly placed sources in a tree, higher level paths (those closer to the root) will become the bottleneck and tend to dominate response times. Once these links become heavily loaded or overloaded, packets will start to be buffered or dropped.

We have addressed the resilience issue of tree-based systems in previous work [5] through the introduction of *co-leaders* and the reliance on *triggered negative acknowledgements* (NACKs). In this paper, we address the bandwidth limitations of normal tree overlays.

Our approach capitalizes on Leiserson's seminal work on fat-trees [27]. Paraphrasing Leiserson, a fat-tree is like a real tree in that its branches become thicker the closer we get to the root, thus overcoming the "root bottleneck" of a regular tree. Figure 1 shows a schematic example of a binary fat-tree. We propose to organize participant end-systems in a tree that closely resembles a Leiserson fat-tree by dynamically placing higher degree nodes (nodes with higher bandwidth capacity) close to the root and increasing the cluster sizes as one ascends the tree.

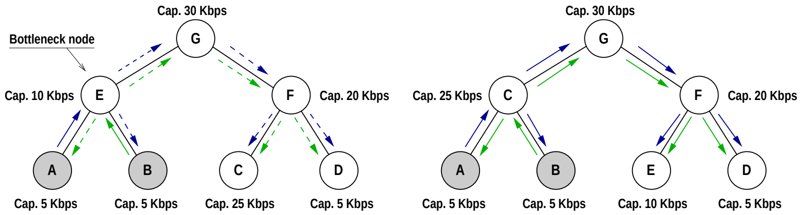
This paper makes three main contributions. First, we introduce the use of Leiserson fat-trees for application-layer multi-source multicast, overcoming the inherent bandwidth limitations of normal tree-based overlay structures (Section 2). Second, after reviewing some background material in Section 3, we describe the design and implementation of *FatNemo*, a new application-layer multicast protocol that builds on this idea (Section 4). Lastly, we evaluate the FatNemo design in simulation, illustrating the benefits of a fat tree approach compared to currently popular approaches to application-layer multicast (Sections 5 and 6). We review some related work in Section 7 and present our conclusion and directions of future work in Section 8.

2 Fat-Trees and the Overlay

The communication network of a parallel machine must support global collective communication operations in which all processors participate [26]. These operations have a wide range, including reduction and broadcast trees, and neighbor exchange. All-to-all personalized communication [21], in which each processor sends a unique message to every other processor, is key to many algorithms. To support such operations well, the network should have (1) minimal and scalable diameter, and (2) maximal and scalable

[1] Measurement studies of widely used application-layer/peer-to-peer systems have reported median session times ranging from an hour to a minute [7,20,34,12].

[2] The access link of an end system becomes its bandwidth bottleneck, thus we can model the bandwidth capacity as a property of the end-system.



(a) Normal tree.

(b) Fat-tree.

Fig. 1. Two binary trees with nodes A and B as sources, publishing at 5 Kbps each. On the left, a normal binary tree where node E becomes the bottleneck, resulting on a reduced (dash line) outgoing stream quality. Node E has to forward the stream A to node B and node G, as well as stream B to node A and node G, thus it needs an outgoing bandwidth capacity of 20 Kbps. However, it has only 10 Kbps available, making it a bottleneck in the tree. On the right, a fat-tree with higher capacity nodes placed higher in the tree.

bisection bandwidth. These goals are very similar to those of a multisource multicast overlay, which can be thought of as providing many-to-many personalized communication, a subset of all-to-all personalized communication. We expect a multisource multicast overlay to have a latency between end-systems that grows only slowly or not at all as the number of end-systems grows. Similarly, we expect the aggregate throughput of the overlay to grow linearly as end-systems are added.

In his seminal work on fat-trees [27], Leiserson introduced a universal routing network for interconnecting the processors of a parallel supercomputer, where communication can be scaled independently of the number of processors. Based on a complete binary tree, a fat-tree consists of a set of processors, located at the leaves, and interconnected by a number of switching nodes (internal to the tree) and edges. Each edge in the tree corresponds to two unidirectional *channels* connecting a parent with each of its children. Channel consist of a bundle of *wires*, and the number of wires in a channel is called its *capacity*. The capacity of the channels of a fat-tree grows as one goes up the tree from the leaves, thus allowing it to overcome the “root bottleneck” of a regular tree. Since their introduction, fat-trees have been successfully applied in massively parallel systems [28,22] as well as in high performance cluster computing [23].

In this paper we propose to organize the end-systems participant in a multicast group, in an overlay tree that closely resembles a Leiserson fat-tree. In common with Leiserson, our goal is to minimize the mean and standard deviation of inter-node communication performance with multiple potential sources.

Emulating fat-trees in an overlay entails a number of challenges such as handling the high level of transiency of end-system populations and addressing their degree of heterogeneity. A straightforward way of approximating a fat-tree is placing those nodes with higher bandwidth capacity[3] closer to the root. Since interior nodes are involved in most inter-node communication paths they strongly influence the overall end-to-end delay and can soon become bottlenecks as one increases the number of sources. Figure 1 shows a schematic example of both a regular binary tree with two 5 Kbps sources (A and B) and a bottleneck node (E) unable to keep up with the publishing rate of the nodes downstream.

Available bandwidth can differ significantly from bandwidth capacity over time, due typically to competing traffic, and any algorithm that attempts to emulate fat-trees in an overlay needs to take account of such dynamism. Also, per-path characteristics must be taken into consideration. Since end-to-end latency is an important factor in the performance of interactive applications, the latency of each link in the overlay, the processing time at each node, and the number of intermediate nodes should be considered carefully. When selecting among possible parents, a closer node may be a better candidate, if it is able to support the load, than an alternative node offering higher available bandwidth. Finally, the mean time to failure of end-systems is significantly lower than for routers, possibly resulting in long interruptions to the data stream as the failure is discovered and the distribution tree repaired.

Although overlay fat-trees can be built above most tree-based multicast systems, in this paper we consider their implementation using Nemo, a high-resilience, latency-optimized overlay multicast protocol [5]. The following section provides some background material on overlay multicast in general and on the operational details of Nemo, before describing the FatNemo design in Section 4.

3 Background

All peer-to-peer or application-layer multicast protocols organize the participating peers into (1) a control topology for group membership related tasks, and (2) a delivery tree for data forwarding. Available protocols can be classified according to the sequence adopted for their construction [1,13]. In a tree-first approach [19,24,32], peers directly construct the data delivery tree by selecting their parents from among known peers. Additional links are later added to define, in combination with the data delivery tree, the control topology. With a mesh-first approach [13,11], peers build a more densely connected graph (mesh) over which (reverse) shortest path spanning trees, rooted at any peer, can be constructed. Protocols adopting an implicit approach [2,10,33,39,35] create only a control topology among the participant peers. Their data delivery topology is implicitly determined by the defined set of packet-forwarding rules.

FatNemo builds on Nemo to emulate a fat-tree; thus, it inherits the latter's high scalability and resilience. In the following paragraphs, we provide a summarized description of Nemo; for more complete details we direct the reader to our previous work [5].

Nemo

Nemo follows the implicit approach to building an overlay for multicasting. The set of communication peers are organized into clusters based on network proximity, where

[3] The maximum outgoing bandwidth that the node is capable of, the capacity of the IP link attaching the node to the network.

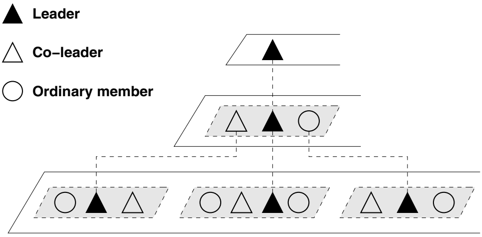


Fig. 2. Nemo's logical organization. The shape illustrates only the role of a peer within a cluster: a leader of a cluster at a given layer can act as leader, co-leader, or an ordinary member at the next higher layer.

every peer is a member of a cluster at the lowest layer. Clusters vary in size between d and $3d - 1$, where d is a constant known as the *degree*. Each of these clusters selects a *leader* that becomes a member of the immediately superior layer. In part to avoid the dependency on a single node, every cluster leader recruits a number of co-leaders to form a supporting crew. The process is repeated at each new layer, with all peers in a layer being grouped into clusters, crew members selected, and leaders promoted to participate in the next higher layer. Hence peers can lead more than one cluster in successive layers of this logical hierarchy. Figure 2 illustrates the logical organization of Nemo.

A new peer joins the multicast group by querying a well-known special end-system, the rendezvous point, for the identifier of the root node. Starting there and in an iterative manner, the incoming peer continues: (i) requesting the list of members at the current layer from the cluster's leader, (ii) selecting from among them who to contact next based on the result from a given cost function, and (iii) moving into the next layer. When the new peer finds the leader with minimal cost at the bottom layer, it joins the associated cluster.

Member peers can leave Nemo in a graceful manner (e.g. user disconnects) or in an ungraceful manner (unannounced, e.g. when the end-system crashes). For graceful departures, since a common member has no responsibilities towards other peers, it can simply leave the group after informing its cluster's leader. On the other hand, a leader must first elect replacement leaders for all clusters it owns before it leaves the session.

To detect unannounced departures, Nemo relies on heartbeats exchanged among the cluster's peers. Unreported members are given a fixed time interval before being considered dead, at which point a repair algorithm is initiated. If the failed peer happens to be a leader, the tree itself must be fixed, the members of the victim's cluster must elect the replacement leader from among themselves.

To deal with dynamic changes in the underlying network, every peer periodically checks the leaders of the next higher layers and switches clusters if another leader has a lower cost (i.e. lower latency) than the current one. Additionally, in a continuous process of refinement, every leader checks its highest owned cluster for better suited leaders and transfers leadership if such a peer exists.

Nemo addresses the resilience issue of tree-based systems through the introduction of co-leaders. Co-leaders improve the resilience of the multicast group by avoiding depen-

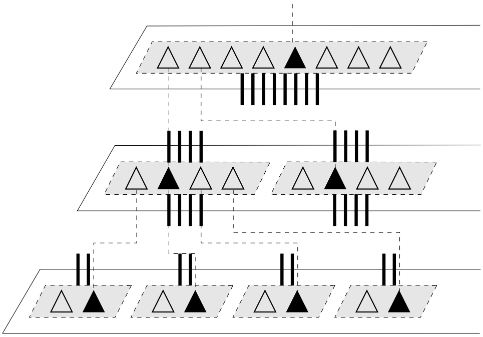


Fig. 3. FatNemo's Topology. The figure illustrates how the tree gets fatter when moving toward the root. This tree has a cluster degree of 2.

dencies on single nodes and providing alternative paths for data forwarding. In addition, crew members share the load from message forwarding, thus improving scalability.

4 FatNemo Design

To build an overlay fat-tree, FatNemo relies on three heuristics: (1) higher bandwidth degree nodes should be placed higher up in the tree, (2) all peers must serve as crew members in order to maximize load balancing, and (3) the size of clusters should increase exponentially as one ascends the tree. The following paragraphs provide the motivations behind each of these heuristics.

The per-node bandwidth constraint is critical for bandwidth-demanding applications and can be stated as the number of full-rate streams a peer is able to support, i.e. its *out-degree*. By organizing peers based on their out-degrees [36,13], we intend to reduce the bandwidth constraints of links higher up the tree. Since the process of estimating available bandwidth is time consuming, peers initially join the tree based on proximity. Once in the tree, every leader checks its highest owned cluster for better suited leaders in terms of bandwidth availability, and transfers leadership if such a peer exists. This process assures that high out-degree peers will gradually ascend to higher layers, thus transforming the tree into a bandwidth optimized fat-tree.

In traditional fat-trees the number of wires increases as one ascends the tree. Considering an overlay unicast connection as a "wire", the number of wires in FatNemo increases together with the crew size as one moves toward the root – the maximum possible number of wires is thus achieved by setting the crew size equal to the cluster size. The size of a cluster at layer i in FatNemo varies between d_i and $2d_i + 2$, and grows exponentially ($d_i = d_0^{i+1}$) as we move up the layers. The 0-th layer contains the leaf nodes and has a degree d_0 of 3 (same as Nice and Nemo). The increased number of wires helps avoid higher level links from becoming the bottleneck of the system, as alternate paths share the load and reduce the forward responsibility of each peer.

Table 1. Cluster and Crew Size as a function of the cluster degree d, for a 20,000 peer population. The variable x is a place holder for the cluster index starting at 0 for the lowest layer.

Protocol	Cluster Size $k(x)$		Crew Size $c(x)$
Nice	$d \dots 3d - 1$	$d = 3 : 3 \dots 8$	1
Nemo	$d \dots 3d - 1$	$d = 3 : 3 \dots 8$	3
FatNemo	$d^{x+1} \dots 2d^{x+1} + 2$	$d = 3, x = 1 : 9 \dots 20$	$k(x)$

Beyond increasing the number of wires, large crew sizes also help reduce the depth of a tree (when compared with its constant cluster-sizes equivalent). A smaller depths means a lower total number of end-system hops, and should translate in a reduction on the end-to-end delay.

Figure 3 illustrates how FatNemo constructs a fat-tree. In this simple example $d_0 = 2$, so clusters scale up by a factor of 2 as we ascend the tree. Notice that these links/wires are indeed $(d_{i+1} \dots 2d_{i+1} + 2)$ to $(d_i \dots 2d_i + 2)$ relations, as every crew member of the next higher layer will talk to the crew members of the immediately lower layer. For clarity in the graph, this set of links is represented in the graph by d_i lines.

To better understand the positive effect of FatNemo's heuristics, we show an instantiation of them with a population of $20,000$ peers from a popular on-line game. [4]

To begin, let's generalize the concept of *out-degree*. The out-degree of a peer, d_{out}, is equivalent to the total forwarding responsibility of a node, and it can be stated as a function of the number of layers L in which the node participates, the cluster size $k(x)$ and the crew size $c(x)$ at layer x (Equation (1)). Table 1 illustrates the parameter values for FatNemo and two alternative protocols.

$$d_{out} = \sum_{x=0}^{L-1} \frac{k(x) - 1}{c(x)} \tag{1}$$

Now, to calculate the requirements for the root node, we must first estimate the number of layers for a given protocol which is a function of the number of peers in a cluster [5]. Nice and Nemo have, in expectation, 5.5 nodes per cluster at every layer. Using this value as an approximation for the cluster size, a traditional tree for this population size will be about 7 layers in depth. FatNemo, on the other hand, has a variable expected number of nodes per cluster, and its expected tree depth is 4 layers.

Based on the expected depth of the different trees for this example population, we can now calculate the out-degree requirements on their root nodes. According to the generalized out-degree equation introduced in the previous paragraph (Equation (1)), Nice requires a root out-degree of 31.5, or almost three times more than what is needed from a Nemo's root (10.5) with a crew size of 3. In other words, the root of a traditional

[4] The number corresponds to the active populations of players in *hattrick.org*, an online soccer game.

[5] The average number of peers in a cluster is equal to the mean cluster size, which can be computed as the mean of the low and high cluster boundary.

tree for a 20,000 peer population must support 31.5 times the source rate to fulfill its forwarding responsibility! By emulating a fat-tree in the overlay, FatNemo avoids this "root bottleneck" requiring only a root with an out-degree of 3.7.

5 Evaluation

We analyze the performance of FatNemo through simulation and compare it to that of three other protocols – Narada [14], Nice [2] and Nice-PRM [3]. We evaluate the effectiveness of the alternative protocols in terms of performance improvements to the application and protocol's overhead, as captured by the following metrics:

Response Time: End-to-end delay (including retransmission time) from the source to the receivers, as seen by the application. This includes path latencies along the overlay hops, as well as queueing delay and processing overhead at peers along the path. A lower mean response time indicates a higher system responsiveness, and a smaller standard deviation implies better synchronization among the receivers.

Delivered Packets: Number of packets successfully delivered to all subscribers within a fixed time window. It indirectly measures the protocol's ability to avoid bottlenecks in the delivery tree.

Delivery Ratio: Ratio of subscribers that have received a packet within a fixed time window. Disabled receivers are not accounted for.

Duplicate Packets: Number of duplicate packets per sequence number, for all enabled receivers, reflecting an unnecessary burden on the network. Packets arrived outside of the delivery window are accounted for as duplicates, since the receiver already assumed them as lost.

Control-Related Traffic: Total control traffic in the system, in mega bits per second (Mbps); part of the system's overhead. We measure the total traffic during the observation interval by accounting packets at the router level. A network packet traversing four routers (including the source and destination node) will account as three sent packets, one for every router which has to forward it.

The remainder of this section briefly discusses implementation details of the compared protocols and describes our evaluation setup. Section 6 presents our evaluation results.

5.1 Details on Protocol Implementations

For each of the three alternative protocols, the values for the available parameters were obtained from the corresponding literature [13,2,3].

For Narada [13], we employ the bandwidth-only scheme for constructing the overlay, as this will result in maximum throughput. For Nice [2] and Nice-PRM [3], the cluster degree, k, is set to 3. We use Nice-PRM(3,0.02) with three random peers chosen by each node, and with two percent forwarding probability.

For FatNemo, the cluster degree at the lowest layer is set to three. Cluster degree grows exponentially with every layer, being nine in the second lowest layer, 27 in the third, and so on.

Our implementations of the alternative protocols closely follow the descriptions from the literature, and have been validated by contrasting our results with the published values. However, there are a number of improvements to the common algorithms, such as the use of periodic probabilistic maintenance operations, that while part of FatNemo, were made available to all protocols in our evaluations. The benefits from these algorithms help explain the performance improvements of the different protocols with respect to their original publications [13,2,3].

5.2 Experimental Setup

We performed our evaluations through detailed simulation using SPANS, a locally written, packet-level, event-based simulator. Simulations were run using GridG [29,30] topologies with 5510, 6312 and 8115 nodes, and a multicast group of 256 members. GridG leverages Tiers [18,8] to generate a three-tier hierarchical network structure, before it applies a power law enforcing algorithm that retains the hierarchical structure.

Members were randomly associated with end systems, and a random delay of between 0.1 and 80 ms was assigned to every link. The links use drop-tail queues with a buffer capacity of 0.5 sec. We configured GridG to use different bandwidth distributions for different link types [25]. We assume that the core of the Internet has higher bandwidth capacities than the edge, as shown in Fig. 4. In all three scenarios, the bandwidth has a uniform distribution with ranges shown in Fig. 4.

Scenario	Routers	End systems	Links	Client-Stub	Stub-Stub	Transit-Stub	Transit-Transit
Low-B/W	510	5000	11240	400-6000	3000-8000	4000-10000	10000-20000
Medium-B/W	312	6000	12730	800-8000	4000-10000	6000-15000	15000-30000
High-B/W	615	7500	16450	1000-15000	10000-30000	10000-50000	50000-100000

Fig. 4. Three simulation scenarios: Low-, Medium- and High-Bandwidth. Bandwidth is expressed in Kbps.

Each simulation experiment lasted for 500 sec. (simulation time). All peers join the multicast group by contacting the rendezvous point at uniformly distributed, random times within the first 100 sec. of the simulation. A warm-up time of 200 sec. is omitted from the figures. Publishers join the network and start publishing at the beginning of the simulation. Starting at 200 sec. and lasting for about 300 sec., each simulation has a phase with membership changes. We exercise each protocol with and without host failures during this phase. Failure rates are set based on those obtained from a published report of field failures for networked systems [37]. Nodes fail independently at a time sampled from an exponential distribution (with *mean time to failure* (MTTF) equal to 60 min.) and rejoin shortly after (time sampled from an exponential distribution with *mean time to repair* (MTTR) equal to 10 min.). The two means were chosen asymmetrically to allow, on average, 6/7 of all members to be up during this phase. The failure event sequence was generated a priori based on the above distribution and used for all protocols and all runs.

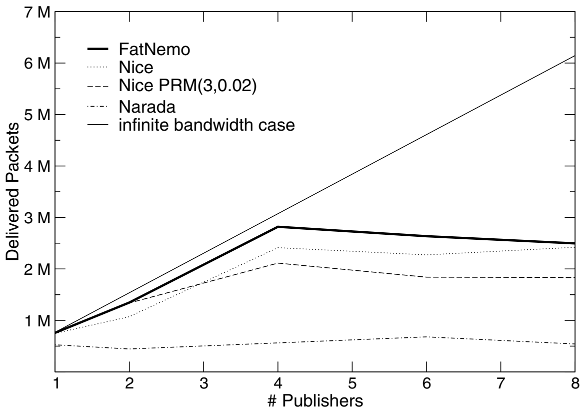


Fig. 5. Delivered packets (256 end hosts, Low-Bandwidth scenario).

In all experiments, we model multi-source multicast streams to a group. Each source sends constant bit rate (CBR) traffic of 1000 Byte payload at a rate of 10 packets per second. The buffer size was set to 16 packets, which corresponds to the usage of a 1.6-second buffer, a realistic scenario for applications such as video conferencing.

6 Experimental Results

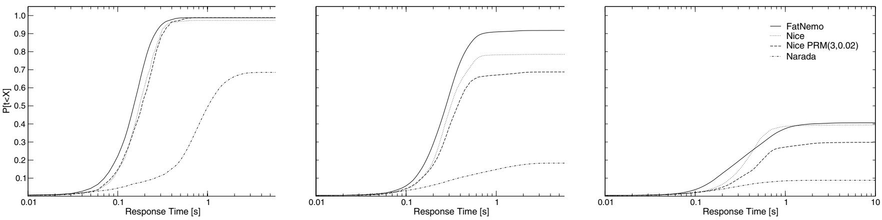
This section presents early evaluation results of FatNemo and compares them with those of three alternative protocols. The reported results are from five runs per protocol obtained with the different GridG topologies and the Low-Bandwidth scenario. Similar results were obtained with the Mid- and High-Bandwidth scenarios.

Figure 5 shows the average number of delivered packets of all runs with no host failures. As we increase the number of publishers, the protocol's data delivery topology collapses. This happens first for Narada, which is unable to handle the full publishing rate from one publisher. Nice and Nice PRM handle an increasing number of publishers better; however, they deliver substantially fewer packets when compared with FatNemo. FatNemo is best at avoiding bottlenecks in the delivery tree, delivering the most packets when the network is overloaded (as seen with 8 publishers).

The performance of a multi-source multicast system can be measured in terms of mean and standard deviation of the response time. Table 2 shows these two metrics for the evaluated protocols with one publisher. FatNemo outperforms Nice, Nice PRM and Narada in terms of mean and standard deviation of response time. With an increased number of publishers the relative number of delivered packets for Nice, Nice PRM and Narada decreases compared to FatNemo, which makes it impossible to fairly compare the response time for more than one publisher based only on one number. The problem stems from that fact that, when lowering its delivery ratio a protocol will drop those

Table 2. Response Time (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	Mean	Std
FatNemo	0.158	0.073
Nice	0.183	0.082
Nice-PRM(3,0.02)	0.195	0.086
Narada	0.770	0.464



(a) 1 Publisher

(b) 4 Publisher

(c) 8 Publisher

Fig. 6. Response Time CDF with 1, 4 and 8 publishers (256 end hosts, Low-Bandwidth scenario).

packets with high response time more likely than others. Thus, comparing response times across protocols with significantly different delivery ratios under stress will give less resilient protocols an unfair advantage.

Figure 6.(a) shows the Cumulative Distribution Function (CDF) of the response time per packet for one publisher. The y-axis is normalized to the infinite bandwidth case, i.e. when all receivers receive all possible packets. FatNemo, Nice and Nice PRM perform well, but FatNemo's flatter tree results in an improved response time. Narada is only able to deliver a fraction of all possible packets, and only then with a substantially high delay. With increasing number of publishers, the protocols start running into bottlenecks. Despite the harder conditions, FatNemo is able to outperforms the alternative protocols in terms of packet delivery times as illustrated in Fig. 6.(b) and Fig. 6.(c).

Table 3 shows the delivery ratio using one publisher with and without end-systems failures. We see that FatNemo performs as well as Nice PRM under a low-failure rate scenario with only a drop of 2.1% in delivery ratio. Nice has a slightly lower delivery

Table 3. Delivery Ratio (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	No Failures	With Failures
FatNemo	0.987	0.966
Nice	0.973	0.956
Nice-PRM(3,0.02)	0.989	0.970
Narada	0.685	0.648

Table 4. Overhead (1 Publisher, 256 end hosts, Low-Bandwidth scenario).

Protocol	Duplicate packets	Control Traffic [Mbps]
FatNemo	0.367	17.99
Nice	0.000	9.211
Nice-PRM(3,0.02)	5.168	11.48
Narada	0.006	157.3

ratio, while Narada suffers already from a collapsed delivery tree with only about 70% delivery ratio. In general, the delivery ratio will decrease as the number of publishers increases, as the protocol's data delivery topology slowly collapses.

The overhead of a protocol can be measured in terms of duplicate packets. We show this metric in the second column of Table 4. Despite its high delivery ratio, FatNemo incurs, in average, only 0.367 duplicate packets per sequence number, while Nice-PRM suffers from 5.168 duplicate packets per sequence number generated by its probabilistic forwarding algorithm. Nice and Narada feature almost no duplicates, but at a high cost in term of delivery ratio as shown in Table 3. FatNemo's control related traffic is higher than for Nice and Nice-PRM, a result of its larger cluster cardinality higher in the tree. The control traffic is accounted for at router level, thus the choice of a peer's neighbors in FatNemo also adds additional overhead, as it opts not for the closest, but for the peer with highest bandwidth.

7 Related Work

Numerous protocols have been proposed to address the demand for live streaming applications. One of the first end-system multicast protocol was Narada [13], a multisource multicast system designed for small to medium sized multicast groups. Peers in Narada are organized into a mesh with fixed out-degree, with every peer monitoring all others to detect end-system failures and network partitions. The per-source multicast tree is built on top of this mesh from the reverse shortest path between each recipient and the source. Since the tree construction algorithm does not account for cross traffic, a powerful link is likely to be used by many multicast links, limiting the efficiency of the multicast system. FatNemo uses crew members to share the forwarding load, thus relaxing the burden on a single high bandwidth path. Overcast [24] organizes dedicated servers in a single-source, bandwidth optimized, multicast tree. In contrast, FatNemo is an end-system overlay that constructs a global optimized fat-tree for multisource multicasting. Banerjee et al. [2] introduce Nice and demonstrate the effectiveness of overlay multicast across large scale networks. The authors also present the first look at the robustness of alternative overlay multicast protocols under group membership changes. FatNemo adopts the same implicit approach, and its design draws on a number of ideas from Nice such as its hierarchical control topology. FatNemo introduces co-leaders to improve the resilience of the overlay and adopts a periodic probabilistic approach to reduce/avoid the cost of membership operations.

A new set of projects have started to address the resilience of overlay multicast protocols [3,9,35,5,31,38]. ZigZag [35], a single-source protocol, explores the idea of splitting the control and data delivery task between two peers in each level, making both responsible for repairs under failures. With PRM [3] Banerjee et al. propose the use of probabilistic forwarding and NACK-based retransmission to improve resilience. In order to reduce the time-to-repair, Yang and Fei [38] argue for proactively, ahead of failures, selecting parent replacements. CoopNet [31] improves resilience by building several disjoint trees on a centralized organization protocol and employing Multiple Description Coding (MDC) for data redundancy. Nemo [5] and FatNemo build redundancy into the overlay through co-leaders; different from the previously described protocols, they make all crew members share forwarding responsibilities while all cluster members are in charge of repair operations. These simple measures enable an uninterrupted service to downstream peers especially during recovery intervals. We are currently exploring the use of data redundancy using forward error correction (FEC) encoding [6].

Aiming at bulk data distribution, protocols such as Splitstream [9] , Bittorrent [15] and Bullet [25] have proposed simultaneous data streaming over different paths to better share the forwarding load and increased downloading capacity. The methods differ in how they locate alternate streaming peers. In comparison, FatNemo exploits alternate paths for resilience and load balancing.

8 Conclusions and Further Work

In this paper we introduced the parallel architecture concept of fat trees to overlay multicast protocols. We have described FatNemo, a novel scalable peer-to-peer multicast protocol that incorporates this idea to build data delivery topologies with minimized mean and standard deviation of the response time. Simulation results show that FatNemo can achieve significantly higher delivery ratios than alternative protocols (an increase of up to 360% under high load), while reducing the mean (by up to 80%) and standard deviation (by up to 84%) of the response time in the non-overloaded case. Under a heavy load and a realistic host failure rate, the resulting protocol is able to attain high delivery ratios with negligible cost in terms of control-related traffic. We are currently validating our findings through wide-area experimentation.

Acknowledgments. We are grateful to Janine M. Casler, Kevin Livingston and Ananth Sundararaj for their helpful comments on early drafts of this paper.

References

1. S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols, 2002. Submitted for review.
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, August 2002.
3. S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Proc. of ACM SIGMETRICS*, June 2003.

4. M. Bawa, H. Deshpande, and H. Garcia-Molina. Transience of peers & streaming media. In *Proc. of HotNets-I*, October 2002.
5. S. Birrer and F. E. Bustamante. Nemo - resilient peer-to-peer multicast without the cost. Tech. Report NWU-CS-04-36, Northwestern U., April 2004.
6. R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, 1994.
7. F. E. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in P2P protocols. In *Proc. of IWCW*, October 2003.
8. K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.
9. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM SOSP*, October 2003.
10. M. Castro, A. Rowstron, A.-M. Kermarrec, and P. Druschel. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8), 2002.
11. Y. Chawathe. *Scattercast: an architecture for Internet broadcast distribution as an infrastructure service*. Ph.D. Thesis, U. of California, Berkeley, CA, Fall 2000.
12. Y.-H. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an Internet broadcast system based on overlay multicast. In *Proc. of USENIX ATC*, June 2004.
13. Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication*, 20(8), October 2002.
14. Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, June 2000.
15. B. Cohen. BitTorrent. bitconjurer.org/BitTorrent/, 2001. File distribution.
16. S. E. Deering. Multicast routing in internetworks and extended LANs. In *Proc. of ACM SIGCOMM*, August 1988.
17. C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1), January/February 2000.
18. M. B. Doar. A better model for generating test networks. In *Proc. of Globecom*, November 1996.
19. P. Francis. Yoid: Extending the Internet multicast architecture. http://www.aciri.org/yoid, April 2000.
20. K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP*, December 2003.
21. S. Hinrichs, C. Kosak, D. O'Hallaron, T. Stricker, and R. Take. An architecture for optimimal all-to-all personalized communication. In *Proceedings of the 6th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 310–319, 1994.
22. M. Homewood and M. McLaren. Meiko CS-2 interconnect elan – elite design. In *IEEE Hot Interconnects Symposium*, August 1993.
23. InfiniBand Trade Association. Infiniband architecture specification (1.0.a). www.infinibandta.com, June 2001.
24. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable multicasting with and overlay network. In *Proc. of the 4th USENIX OSDI*, October 2000.
25. D. Kostić, A. R. adn Jeannie Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of the 19th ACM SOSP*, October 2003.
26. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

27. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.
28. C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S.-W. Yang, and R. Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.
29. D. Lu and P. A. Dinda. GridG: Generating realistic computational grids. *ACM Sigmetrics Performance Evaluation Review*, 30(4):33–41, March 2003.
30. D. Lu and P. A. Dinda. Synthesizing realistic computational grids. In *Proc. of SC2003*, November 2003.
31. V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *Proc. of IEEE ICNP*, 2003.
32. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of USENIX USITS*, March 2001.
33. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, November 2001.
34. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of USENIX ATC*, December 2004.
35. D. A. Tran, K. A. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE INFOCOM*, April 2003.
36. Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proc. of IEEE GlobeCom*, November 1995.
37. J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT system field failure data analysis. In *Proc. of PRDC*, December 1999.
38. M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *Proc. of IEEE INFOCOM*, March 2004.
39. S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSSDAV*, June 2001.